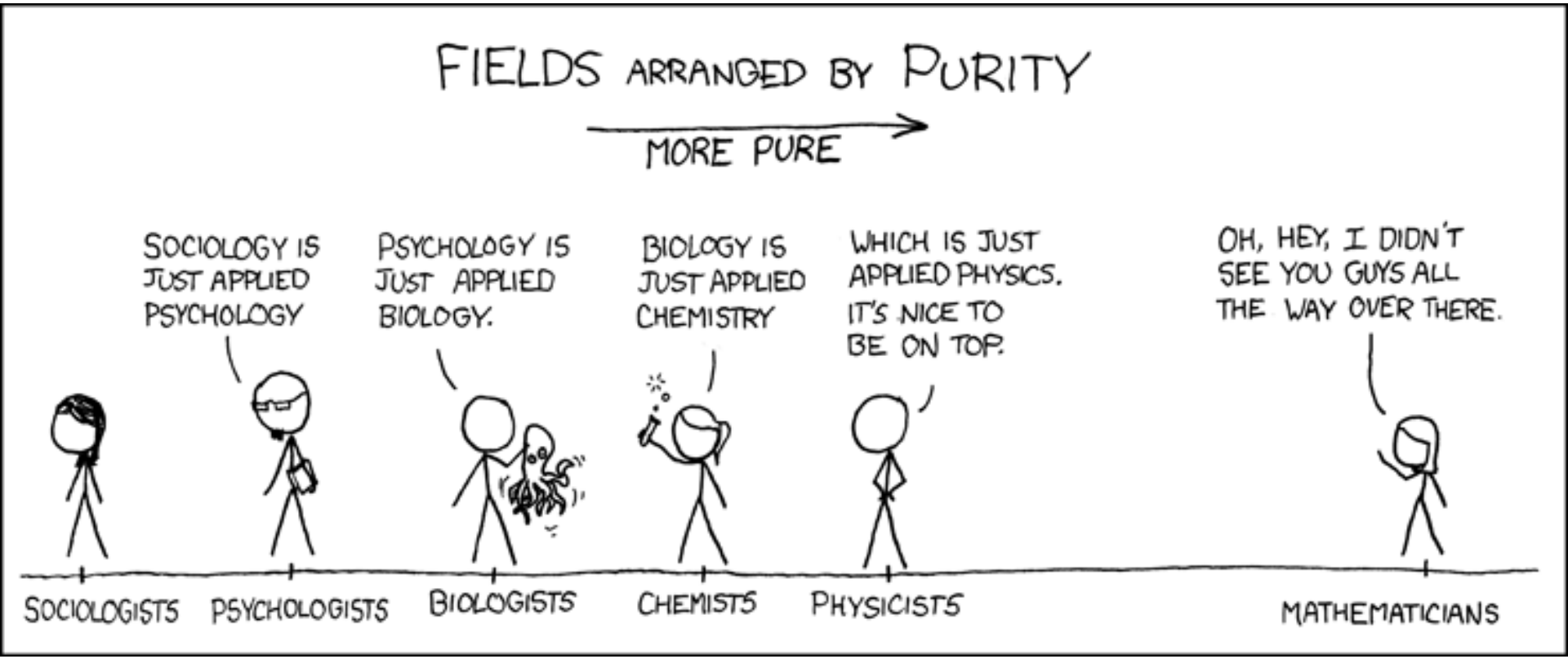
Sissejuhatus infotehnoloogiasse

Lahenduvus


FIELDS ARRANGED BY PURITY
MORE PURE
SOCIOLOGY IS JUST APPLIED PSYCHOLOGY
PSYCHOLOGY IS JUST APPLIED BIOLOGY.
BIOLOGY IS JUST APPLIED CHEMISTRY
WHICH IS JUST APPLIED PHYSICS. IT'S NICE TO BE ON TOP.
OH, HEY, I DIDN'T SEE YOU GUYS ALL THE WAY OVER THERE.
SOCIOLOGISTS
PSYCHOLOGISTS
BIOLOGISTS
CHEMISTS
PHYSICISTS
MATHEMATICIANS

Loengu ülevaade

- Eelmine loeng**: keerukus** ehk kui ruttu ülesannet lahendada saab?

- See loeng**: lahenduvus** ehk kas üldse ülesannet lahendada saab?
  - Probleem
  - Põhi-ideed
  - Põhimõisted
  - Näited

Algoritmide omadustest

Mõned olulised algoritmide omadused, mida eeldame / soovime:

- Kindlasti:
  - **Deterministic:** Given the same input, it produces the same output.
  - **Finite**: It can be described in a finite number of steps
  - **Definite**: Each step has a clearly defined meaning.
- Soovitavalt:
  - **Correct:** It produces correct answers
  - **Time Bounded**: It eventually stops
  - **Fast:** it not only stops, but stops quickly

Algoritmide omadustest

Mõned olulised algoritmide omadused, mida eeldame / soovime:

- Kindlasti:
  - **Deterministic:** Given the same input, it produces the same output.
  - **Finite**: It can be described in a finite number of steps
  - **Definite**: Each step has a clearly defined meaning.
- Soovitavalt:
  - **Correct:** It produces correct answers
  - **Time Bounded**: It eventually stops
  - **Fast:** it not only stops, but stops quickly

Verifitseerimine, testimine

Lahenduvus

Keerukus

Lahenduvus

- Teame, et iga probleemi jaoks ei leidu kiiret algoritmi.
- Kas aga iga probleemi jaoks on üldse olemas algoritmi, mis seda lahendab?
- Eeldame siin, et vaatame ainult probleeme, mis on täpselt ja üheselt kirjeldatud ja kus on lahendamiseks olemas piisavalt infot (a la travelling salesman, malemäng jne)
- **Selgub, et iga täpselt formuleeritud probleemi (matemaatika- ja programmeerimisprobleemid) jaoks ei leidugi lahendavat algoritmi!**
- Vähe sellest: kui võtta “juhuslik” probleem, siis tõenäosus, et lahendav algoritm leidub, on lõpmatult väike!

„lahenduvus“ tavamõttes ...

Tüüpilised põhjused, miks me ei saa **tavaprobleeme** lahendada: näiteks, kuidas saada kiiresti väga palju raha:

- Ei ole piisavalt infot:
  - Kui teaks, kus on mõni peidetud aare, kaevaks kohe üles.
  - Kui teaks, mis firma ülesostmine homme välja kuulutatakse, ostaks selle aktsiaid.
- Juhuslikkus segab:
  - Teel aaret välja kaevama minnes võin kraavi sõita.
  - Kasiinos panustades ei tea, mis number ruletis tuleb.
  - Kas aktsia lähiajal tõuseb või langeb, sõltub tihti hullult paljudest juhuslikest teguritest.

„lahenduvus“ matemaatika mõttes ...

Matemaatika- ja programmeerimisprobleemide juures eeldame lahenduvuse valdkonnas üldjuhul, et

- Infot **on** piisavalt: meil on olemas kõik vajalikud aksioomid / programm / täpne ülesanne, näiteks:
  - Maleseis ja käigureeglid.
  - Täisarvude massiivi sorteerituse kriteerium.
  - Programmi sisend ja seismajäämise tuvastamise kriteerium.
- Juhuslikkust **ei ole**:
  - Malemäng käib täpselt reeglite järgi.
  - Sorteerimisel ei toimu juhuslikke muutusi massiivis.
  - Programm ei tee juhuslikke tegevusi.

Lahenduvus uurimisvaldkonnana

Algoritmi- ehk rekursiooniteooria on suur arvutiteaduse uurimisvaldkond.

Ingliskeelne harilik nimi: algorithm theory, recursion theory

Lahenduvus inglise keeles: decidability või computability

- Uuritakse, millistele ülesannetele on algoritme, millistele ei
- Uuritakse, mis ülesande lahendamine taandub teisele ülesandele
- Uuritakse lahendumist, poollahendumist, kreatiivseid hulki jne jne
- Uuritakse lõpmatuse struktuuri, mis on kirjeldamatult keeruline
- Uuritakse lahendumise struktuuri, mis on kirjeldamatult keeruline
- Uuritakse loogikaklasside lahendumise taandumist muudele ülesannetele
- ....

Intuitiivne seletus lahendamatusele

- Saab näidata, et erinevaid probleeme on lõpmatult rohkem, kui erinevaid algoritme.
- Kuna probleeme on lõpmatult rohkem kui algoritme, siis iga probleemi jaoks lihtsalt “ei jätku” lahendavat algoritmi.
- Meie plaan, kuidas seda näidata:
  - Näitame, et **algoritme on sama palju, kui täisarve** (lihtne)
  - Näitame, et **probleeme on vähemalt sama palju, kui reaalarve** (veidi keerulisem)
  - Näitame, et **reaalarve on lõpmatult rohkem kui täisarve** (Cantori teoreem)

Algoritme on sama palju (või vähem?) kui täisarve

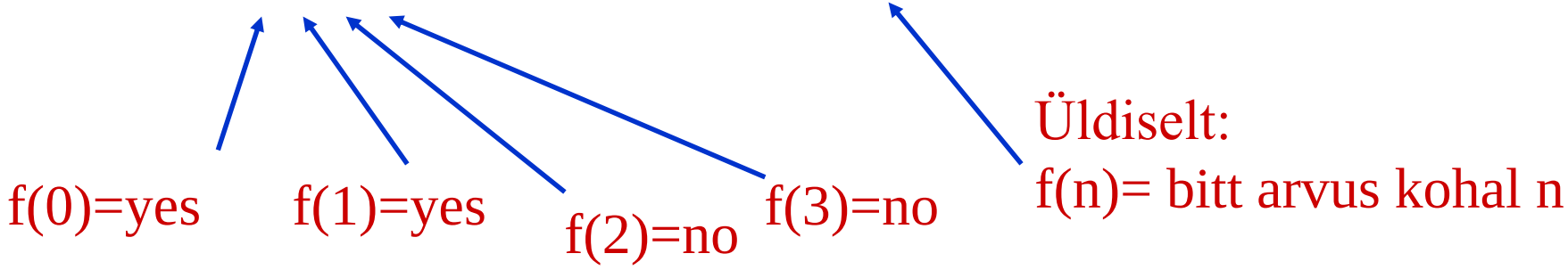
- Iga algoritmi saab kirjutada mistahes programmeerimiskeeles.
- Valime näiteks Pythoni keele.
- **Iga Pythonikeelne programm on tekstifail, st üks pikk string.**
- String on näiteks “asas asss ddddd”.
- String koosneb järjestikustest baitidest, iga bait vahemikus 0-255
- **Iga string vastab ühele täisarvule:** vaatame stringi kui 256-süsteemis arvu, näiteks:
  - Baidid **0 22 64** annavad arvu **22*256+64**
  - Baidid **64 120 68** annavad arvu **64*256 ruudus + 120*256 + 64**
- NB! Iga string ei ole korrektne Pythoni programm. Vastupidi küll.

Probleeme on sama palju kui reaalarve

- **Mis on reaalarv? Arv, kus koma järel võib olla kuitahes palju komakohti.**
- **Näiteks**: 2,23242545344123123 1...
- 2, pi, ¾, ruutjuur kahest on kõik reaalarvud.
- Kahendsüsteemis reaalarvul on iga number kas 0 või 1, näiteks:
  110.11010010110101111010101....

- Võtame ühe spetsiifilise klassi probleemidest: “yes-no” probleemid täisarvudel:
  Algoritm võtab sisendiks täisarvu ja peab vastama “yes” või “no”.

- Kui palju selliseid probleeme on?
- Iga kahendsüsteemis reaalarv alla ühe vastab ühele probleemile:
- `0.11001010101010101010101 ...`



Cantori teoreem: sissejuhatus

- **Reaalarvude hulk on suurem (võimsam) kui positiivsete täisarvude hulk.**
- Enne uurime, kuidas on lugu pos/neg täisarvudega ja murdudega.
- Pos/neg täisarve oleks justkui kaks korda rohkem, kui positiivseid täisarve??

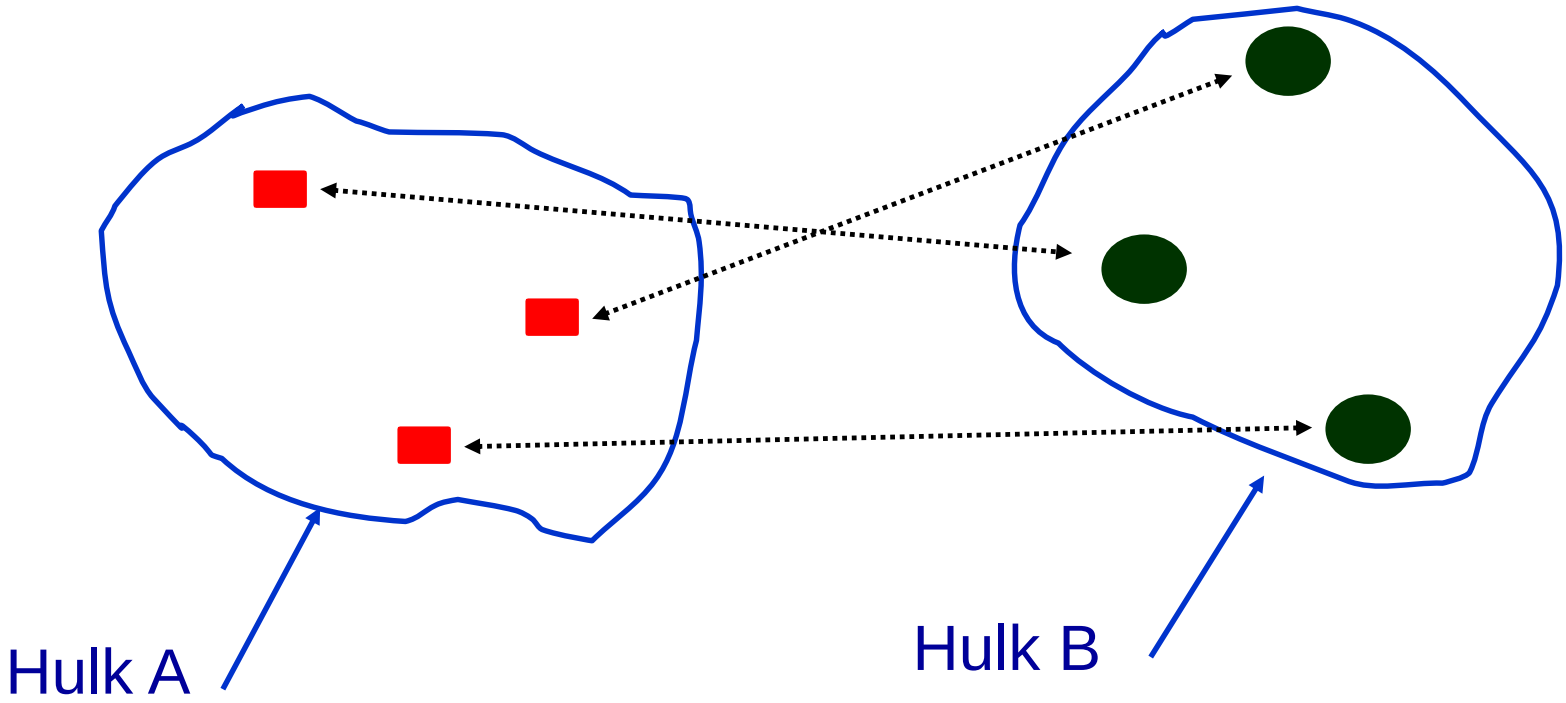
```
... -4   -3   -2   -1    0    1    2    3    4 ...
                        0    1    2    3    4 ...
```

Neg/Pos täisarvud: N

Positiivsed täisarvud: Z

Cantori teoreem: sissejuhatus

- Mida tähendab väide: “hulk A on sama suur (võimas) kui hulk B”?
- Seda, et hulga A kõik elemendid saab seda üksühesesse vastavusse hulga B elementidega:



Igale A elemendile vastab täpselt üks B element ja vastupidi

Cantori teoreem: sissejuhatus

- Kuidas seada üksühesesse vastavusse kahe lõpmatu hulga N ja Z elemendid?

N:	...	-4	-3	-2	-1	0	1	2	3	4 ...
Z:						0	1	2	3	4 ...

- N oleks justkui suurem kui Z?

Cantori teoreem: sissejuhatus

- Kuidas seada üksühesesse vastavusse kahe lõpmatu hulga N ja Z elemendid?

N:	...	-4	-3	-2	-1	0	1	2	3	4 ...
Z:						0	1	2	3	4 ...

- Seame vastavusse hoopis niimoodi:

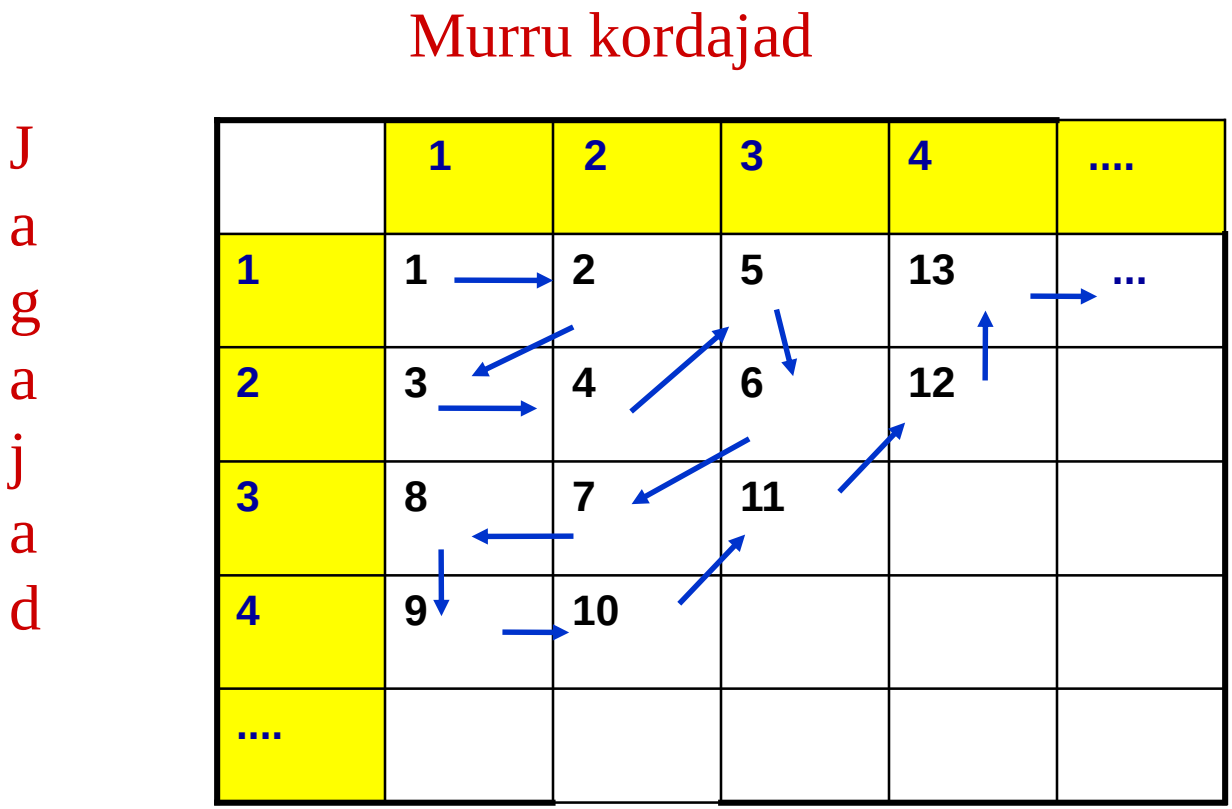
N:	0	1	-1	2	-2	3	-3	4	-4	...
Z:	0	1	2	3	4	5	6	7	8	...

- **Kuna N ja Z saab üksühesesse vastavusse seada, siis on nad sama suured (sama võimsad)!**

Cantori teoreem: sissejuhatus

- Murdarve oleks justkui lõpmatult rohkem, kui positiivseid täisarve?? Samuti vale!

Tegelikult on murdarvud vs pos täisarvud ükskühes vastavuses:



Cantori teoreem: kõigepealt idee

- Koostame kõigi 1-st väiksemate reaalarvude tabeli:

Arvu kohad pärast koma

arvud						
	1	**2**	**1**	**5**	**6**	**...**
	3	**3**	**1**	**0**	**9**	**...**
	2	**8**	**3**	**5**	**6**	**...**
	3	**5**	**6**	**9**	**0**	**...**
					...	

- Ja nüüd konstrueerime diagonaali (**1 3 3 9 ...**) järgi uue arvu, liites diagonaali arvudele ühe (kui tulemus 10, võtame 0): saame **2 4 4 0 ...**

Cantori teoreem: detailselt läbi tehes 1

Arvu kohad pärast koma

arvud

1	**2**	**1**	**5**	**6**	**...**
3	**3**	**1**	**0**	**9**	**...**
2	**8**	**3**	**5**	**6**	**...**
3	**5**	**6**	**9**	**0**	**...**
				...	

- **Konstrueerime uue arvu esimese numbri:**
  - Esimesel real tulbas 1 oli number 1.
  - Meie võtame 1+1 =2
  - Meie uue arvu esimene number on seega 2:
- **Meie uus arv: 0 . 2 ...**

Cantori teoreem: detailselt läbi tehes 2

Arvu kohad pärast koma

arvud						
	1	**2**	**1**	**5**	**6**	**...**
	3	**3**	**1**	**0**	**9**	**...**
	2	**8**	**3**	**5**	**6**	**...**
	3	**5**	**6**	**9**	**0**	**...**
					...	

- **Konstrueerime uue arvu teise numbri:**
  - Teisel real tulbas 2 oli number 3.
  - Meie võtame 3+1 =4
  - Meie uue arvu teine number on seega 4:
- **Meie uus arv: 0 . 2 4 ...**

Cantori teoreem: detailselt läbi tehes 3

Arvu kohad pärast koma

arvud						
	1	**2**	**1**	**5**	**6**	**...**
	3	**3**	**1**	**0**	**9**	**...**
	2	**8**	**3**	**5**	**6**	**...**
	3	**5**	**6**	**9**	**0**	**...**
					...	

- **Konstrueerime uue arvu kolmanda numbri:**
  - Kolmandal real tulbas 3 oli number 3 .
  - Meie võtame 3+1 =4
  - Meie uue arvu kolmas number on seega 4:
- **Meie uus arv: 0 . 2 4 4 ...**

Cantori teoreem: detailselt läbi tehes 4

Arvu kohad pärast koma

arvud

1	**2**	**1**	**5**	**6**	**...**
3	**3**	**1**	**0**	**9**	**...**
2	**8**	**3**	**5**	**6**	**...**
3	**5**	**6**	**9**	**0**	**...**
				...	

- **Konstrueerime uue arvu neljanda numbri:**
  - Neljandal real tulbas 4 oli number 9 .
  - Meie võtame 9+1 = 10, meie võtame siis numbriks 0
  - Meie uue arvu kolmas number on seega 0:
- **Meie uus arv: 0 . 2 4 4 0 ...**

Cantori teoreem: detailselt läbi tehes 5

Arvu kohad pärast koma

arvud						
	1	**2**	**1**	**5**	**6**	**...**
	3	**3**	**1**	**0**	**9**	**...**
	2	**8**	**3**	**5**	**6**	**...**
	3	**5**	**6**	**9**	**0**	**...**
					...	

- Meie uus arv oli: **0 . 2 4 4 0 ...**
- Kas meie uus arv on meie tabelis, st mõni rida tabelis?
- **Igatahes ei saa ta olla esimene rida:**
  - Esimesel real oli esimene arv **1**, meil aga on esimene arv **2**

Cantori teoreem: detailselt läbi tehes 6

Arvu kohad pärast koma

arvud

1	**2**	**1**	**5**	**6**	**...**
3	**3**	**1**	**0**	**9**	**...**
2	**8**	**3**	**5**	**6**	**...**
3	**5**	**6**	**9**	**0**	**...**
				...	

- Meie uus arv oli: **0 . 2 4 4 0 ...**
- Kas meie uus arv on meie tabelis, st mõni rida tabelis?
- **Igatahes ei saa ta olla teine rida:**
  - Teisel real oli teine arv **3**, meil aga on teine arv **4**

Cantori teoreem: detailselt läbi tehes 7

Arvu kohad pärast koma

arvud						
	1	**2**	**1**	**5**	**6**	**...**
	3	**3**	**1**	**0**	**9**	**...**
	2	**8**	**3**	**5**	**6**	**...**
	3	**5**	**6**	**9**	**0**	**...**
					...	

- Meie uus arv oli: **0 . 2 4 4 0 ...**
- Kas meie uus arv on meie tabelis, st mõni rida tabelis?
- **Igatahes ei saa ta olla kolmas rida:**
  - Kolmandal real oli kolmas arv **3**, meil aga on kolmas arv **4**

Cantori teoreem: detailselt läbi tehes 8

Arvu kohad pärast koma

arvud

1	**2**	**1**	**5**	**6**	**...**
3	**3**	**1**	**0**	**9**	**...**
2	**8**	**3**	**5**	**6**	**...**
3	**5**	**6**	**9**	**0**	**...**
				...	

- Meie uus arv oli: **0 . 2 4 4 0 ...**
- Kas meie uus arv on meie tabelis, st mõni rida tabelis?
- **Igatahes ei saa ta olla neljas rida:**
  - Neljandal real oli neljas arv **9**, meil aga on neljas arv **0**

Cantori teoreem: detailselt läbi tehes 9

Arvu kohad pärast koma

arvud

1	**2**	**1**	**5**	**6**	**...**
3	**3**	**1**	**0**	**9**	**...**
2	**8**	**3**	**5**	**6**	**...**
3	**5**	**6**	**9**	**0**	**...**
				...	

- Selline asjade käik ei olnud juhus: me ise konstrueerisime oma arvu!
- Üldiselt on (meie enda konstruktsiooni-meetodi järgi) nii:
- **Igatahes ei saa ta olla N-s rida:**
  - N-ndal real oli N-s arv **X**, meil aga on N-s arv **X+1 (kui X= 9, siis 0)**

Cantori teoreem: detailselt läbi tehes 10

Kokkuvõttes:

- Meie arv 0. 2 4 4 0 .... ei saa olla selles tabelis
- Kui meil oleks tabel kuidagi teisiti tehtud (arvud teises järjekorras) siis **kui me jälle teeksime diagonaali järgi oma uue arvu, siis seda arvu ikka tabelis ei ole.**
- Meie uus arv kahtlemata on reaalarv.
- Seega ei sisalda ükski 1-st väiksemate reaalarvude tabel kõiki reaalarve!

- Mis see siis tähendab:
  - Kõiki reaalarve ei saagi tabelisse panna
  - Iga tabeli N-s rida vastab täisarvule N
  - **Reaalarvude hulk on suurem (võimsam) kui täisarvude hulk**

Cantori teoreemi edasine jätk

Pane tähele, et:

- Iga kahendsüsteemis reaalarv 0 ja 1 vahel **vastab ühele täisarvude alamhulgale**: N-nda biti positsioon ütleb, kas arv N on seal hulgas või ei.

- **Näiteks:**
  - Paarisarvude hulgale vastab: `10101010101010101010....`
  - Kõigi arvude hulgale vastab: `11111111111111111....`
  - Algarvude hulgale vastab: `01010101000101....`

- Cantori teoreem ütleb üldisemalt, et **mingi hulga H kõigi alamhulkade hulk on suurema võimsusega kui see hulk H.**

- **Tekivad lõpmatud ahelad üha suurema võimsusega hulkadest:**
- Täisarvude hulk N1 → N1 alamhulkade hulk N2 → N2 alamhulkade hulk N3 → jne jne ....

Veel üks lahtine matemaatikaprobleem

Kontiinumhüpotees: kas täisarvude lõpmatuse ja reaalarvude lõpmatuse vahepeal on veel lõpmatusi, ehk hulki, mis oleks võimsamad täisarvude omast ja vähem võimsad reaalarvude omast?

On olemas üks suur hulgateooria aksiomaatika ZFC, millest saab tuletada väga suure osa matemaatikast.

On tõestatud, et ei kontiinumhüpoteesi ega tema eitust ei saa järeldada ZFC-st. Mis tähendab, et meil oleks vaja ZFC-le aksioome lisada. Milliseid?

Konkreetne näide mittelahenduvusest: peatuvusprobleem ehk Halting Problem

- Osad programmid peatuvad, osad mitte:

```
def iamhalting(i):  # halts for any i
   while i<10:
      print i*i
      i=i+1

def iamnothalting(i): # does not halt if i<=5
   while i<10:
      print i*i
      if i>5: i=i+1
```

- You plan to write a program that will take in a user's program and inputs and decides whether
  - it will eventually stop, or
  - it will run infinitely in some infinite loop.

Some ideas for writing a halting checker

- Let the checker interpret a given program row-by-row: if the program eventually halts, the checker will detect that.
- How to check that the program **does not halt**?
  - Look for loops such as *while (statement)*
    - If variables in the statements are unchanged e.g.
      - *While (t<1)* with *t* is initialized to *0* but never used in the loop
      - No statements to get out of the loop, such as *goto* or *break*
    - then program will not halt.
  - Add additional checks and methods, each covering some cases and being able to detect non-halting for some programs.
- Can we detect all the reasons why the program does not halt?

Interesting case of a halting problem: 3n+1

```
# Collatz conjecture: seq3np1(i) seems to halt for all i.
# All tries for 1,2,....,huge_number have led to halting.
# But, nobody has found a proof that seq3np1(i) halts for all i.

# Print the 3n+1 sequence from n, halting when it reaches 1.

def seq3np1(n):
  while n != 1:
    print(n)
    if n % 2 == 0:    # n is even
      n = n / 2
    else:             # n is odd
      n = (n * 3) + 1
  print(n)            # the last print is 1
```

Interesting case of a halting problem: 3n+1

```
# It is easy to see that search() will never halt,
# regardless of whether seq3np1(i) halts for all i or not


def search():
  i=1
  while True:
    print "*** checking ",i
    # if seq3np1 does not stop for some i,
    # it will loop forever and never halt
    seq3np1(i)
    i=i+1
    # if the seq3np1 does stop for all i,
    # search() will still not halt
```

Interesting case of a halting problem: 3n+1

```
# Suppose we have a halting solver: Halts(function,param)
# It is easy to see that supersearch will halt if and only if
# seq3np1 does not halt on some i

def supersearch(dummy): # dummy is not actually used
  i=1
  while True:
    print "*** checking ",i
    # regardless whether seq3np1 does or does not stop,
    # the next line will halt:
    if Halts(seq3np1,i):
      i=i+1 # keep searching for non-halting i
    else:
      print "seq3np1 does not halt on",i
      return i
```

Interesting case of a halting problem: 3n+1

```
# If supersearch can be written and works ok, we can compute
# whether seq3np1 halts for all integers, thus solving a major
# open problem

if Halts(supersearch,1): # 1 is just a dummy value
  print "seq3np1 always halts"
else:
  print "seq3np1 does not halt on value"
  print Halts(supersearch,dummy)
```

Proof that a halting checker is impossible: start

- Assume that it is possible to write a program to solve the Halting Problem.
- Denote this program by **HaltAnswerer(*prog*,*inputs*).**

```
HaltAnswerer(prog,inputs):
    if  halts (prog(inputs)):
        return true
    else:
        return false
```

- A program is just a string of characters
  - E.g. your Python program is just a long string of characters
- An input can also be considered as just a string of characters
- So HaltAnswerer is effectively just working on two strings

Proof part 2:

- We can now write another program **Nasty(prog)** that uses **HaltAnswerer** as a subroutine. The program **Nasty(prog)** does the following:

```
Nasty(prog):
    if HaltAnswerer(prog,prog)==true:
        # case 1: Nasty will go into an infinite loop
        while true: x=1
    else:
        # case 2: Nasty will halt
        return
```

- Consider what happens when we run **Nasty(Nasty)**.
- If **Nasty loops infinitely**,
  - HaltAnswerer(Nasty,Nasty) returns false which by [case 2] above means Nasty will halt.
- If **Nasty halts**,
  - HaltAnswerer(Nasty,Nasty) will return true which by [case 1] above means Nasty will loop infinitely.

- Conclusion: Our assumption that it is possible to write a program to solve the halting problem has resulted in a contradiction.

Diagonalization for halting problem: try it out!!

- Each program can be represented by a string and each string can be represented by a natural number

- **Create a table for all programs with a single input where the entry *(i,j)* is**
  - *H* if program *i* halts when program *j* is used as input
  - *NH* if program *i* does not halt when program *j* is used as input

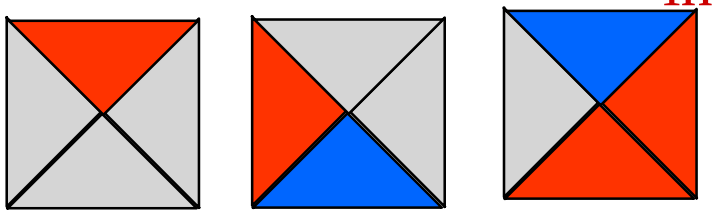
- **Where is *Nasty* in this list?**

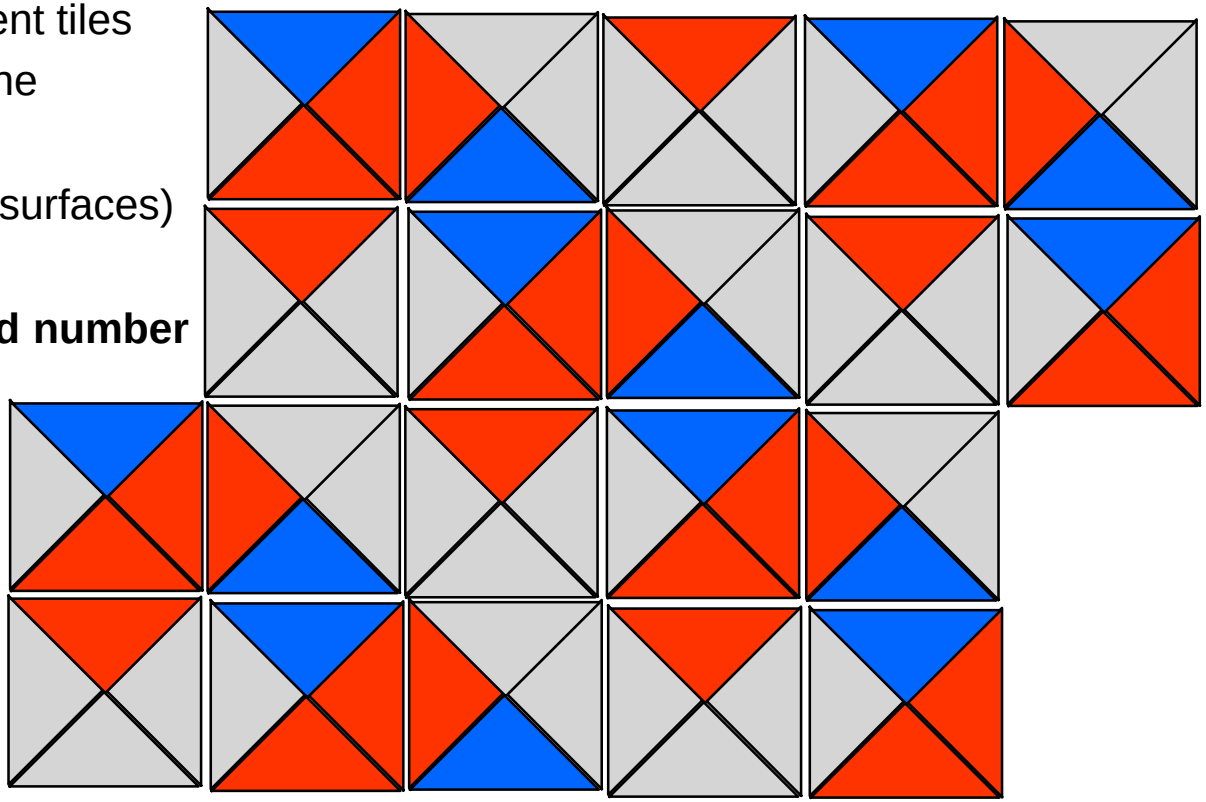
Lihtne näide geomeetriast: “tiling problem”

Assume that you have
a finite set of tiles
(which cannot be rotated)
and you would like
to know if you can tile
an area using those
tiles (adjacent tiles
must have the
same color
on adjoining surfaces)

An **unlimited number of tiles** is available of each kind

Three tiles given in this example

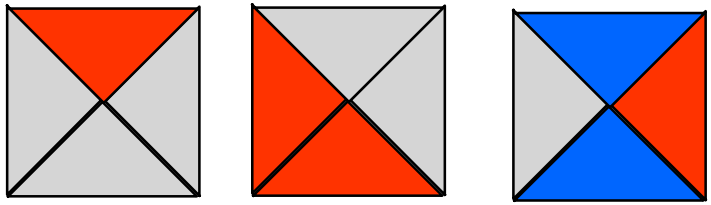


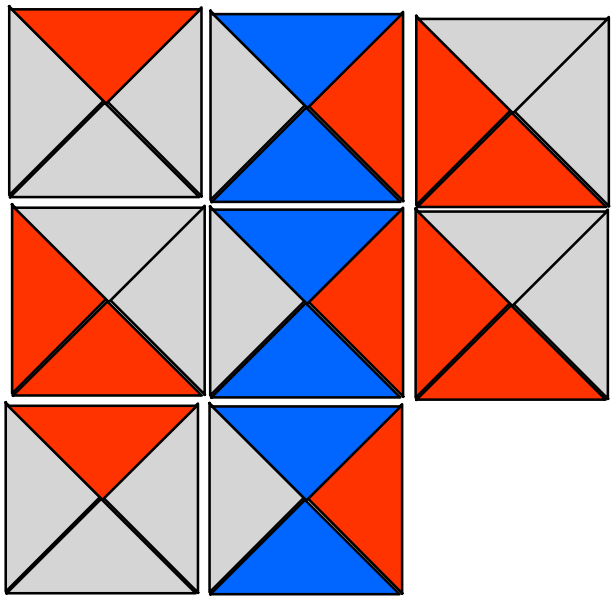


Tiling problem: lõpliku pinna katmine

- **Can I use this set of tiles to tile the floor of a house?**
  - **Lahenduv:** proovime järele kõikvõimalikud paigutused
  - Kuna maja põrand on lõplik, siis proovitavaid variante on lõplik (kuigi väga suur) hulk.
  - Võib meelde jätta, et see on **NP-täieliku keerukusega** ülesanne.
- **Selgub, et mõne plaatide hulgaga saab, mõnega ei!**
  - Lihtne on näiteks juhtum, kus on ainult ühte tüüpi plaate: valge.
  - Aga vaatame järgmist näidet (kolm erinevat plaaditüüpi) , kus ruudukujulise pinna plaatimine ei õnnestu.

Another example tile set:





Error

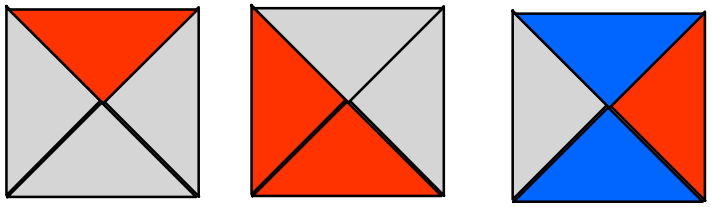
No Tiling for even small areas

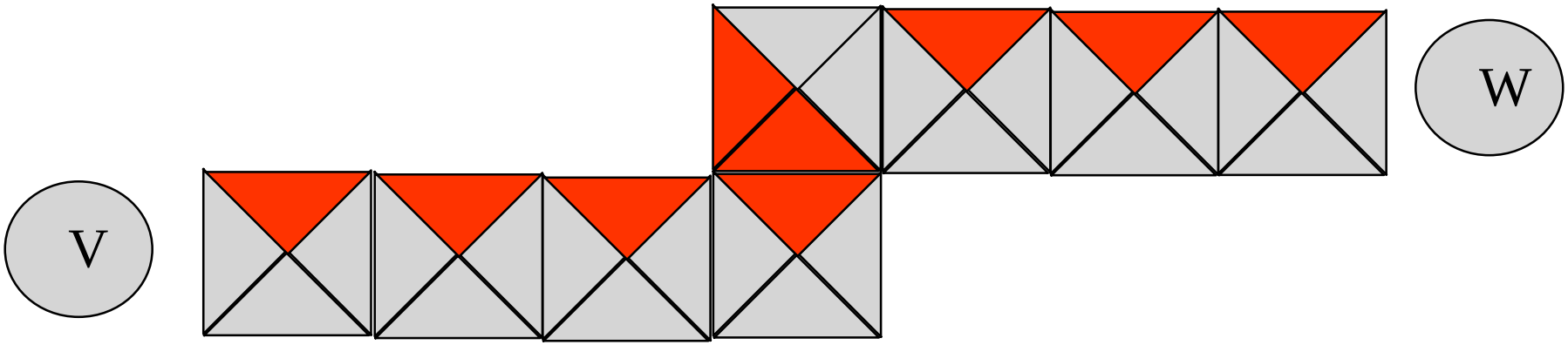
Tiling problem: undecidable question

- **Can a room of any size be tiled using this set of tiles?**
- **St, pind suurusega 2*2, pind 3*3, pind 4*4, jne jne ....**
  - **Ei ole lahenduv!**
  - Pane tähele, et **järeleproovimise meetod** enam ei anna kindlaid tulemusi, sest järgi tuleks proovida lõpmatult palju pinnasuurusi.
  - Pane tähele ka seda, et kui vastus on negatiivne (mingi suurusega N*N pinda ei saa meie plaatidega katta), siis mehaaniline järeleproovimine lõpuks selle avastab.
  - Raskus tekib, *kui tegelikult saab kõiki pindu katta*: järeleproovimise meetod proovib üha suuremaid ja suuremaid pindu, lõpmatuseni.

Analoogiline näide: “Domino Snakes”

- **Is it possible to connect V to W using a tile snake?**





“Domino Snakes” kohta saab tõestada, et:

Lets look at three cases:

- If the plane to lay down the snake in is finite?
  - **trivially decidable**
- If snakes can go anywhere in the plane?
  - **decidable**
- If snakes can only go in half of the plane?
  - **undecidable**

Kuulus näide matemaatikast: Hilberti 10-s probleem

- In 1900, mathematician David Hilbert identified 23 mathematical problems and posed them as a challenge for the coming century.

  https://en.wikipedia.org/wiki/Hilbert%27s_problems

- **The tenth problem asks for an algorithm to test whether a polynomial has an integral root**
- Apparently, Hilbert assumed that such an algorithm must exist.

  https://en.wikipedia.org/wiki/Hilbert%27s_tenth_problem

Probleemi sisu

- Küsimus: kas täisarvuliste kordajate ja astmetega polünoomil on täisarvulised lahendid?
- Kaks näidet:

3x**2 – 2xy – z*y**2 – 7 = 0

lahendid on: x=1, y=2, z=1

x**2 + y**2 + 1 = 0

ei ole täisarvulisi lahendeid

- Eesmärk: leida algoritm, mis iga sellise polünoomi jaoks ütleks kas „on jah täisarvulised lahendid“ või „ei, täisarvulisi lahendeid ei ole“

Kuulus näide matemaatikast: ei ole lahenduv

1970 tõestas **Yuri Matiyasevich,** et alati hakkamasaavat algoritmi täisarvuliste lahendite leidmiseks sellistele polünoomidele ei saa olla.

Poollahenduvus

- Olgu ülesandeks tuvastada, **kas täisarv X kuulub mingisse lõpmatusse täisarvude alamhulka H**.
  - Mõne H jaoks on ülesanne **lahenduv**: näiteks, kui H on paarisarvude hulk, kui H on algarvude hulk jne,
  - Mõne H jaoks ülesanne **ei ole lahenduv**: näiteks, kui H on arvude hulk, millele vastavad programmid peatuvad.

- **Poollahenduvus** tähendab, et kui X juhuslikult kuulub hulka H, siis me saame seda algoritmiga alati näidata. Kui ei kuulu H-i, siis ei saa alati.

- Peatumisprobleemi puhul: paneme X-le vastava programmi käima ja kui ta peatub, siis loomulikult teame, et ta kuulub hulka H
  - Kui ta aga ei peatu, siis meil ei ole kindlat viisi aru saada, et ta ei kuulu hulka H.
  - **Peatumisprobleem on poollahenduv**.

- **On olemas ülesandeid, mis ei ole ka mitte poollahenduvad.**

Lahenduvus: Gödel

Vanad “vist ekslikud” oletused:

[1] Mathematics is consistent. Roughly this means that we cannot prove a statement and its opposite; we cannot prove something horrible like 1=2.

[2] Mathematics is complete. Roughly this means that every true mathematical assertion can be proven i.e. every mathematical assertion can either be proven or disproven.

[3] Mathematics is decidable. This means that for every type of mathematical problem there is an algorithm that, in theory at least, can be mechanically followed to give a solution. We say “in theory” because following the algorithm might take a million years and still be finite

Kurt Gödel

https://de.wikipedia.org/wiki/Datei:1925_kurt_gödel.png

Lahenduvus: Gödel

Hiljem selgus, et:

- In 1930, Kurt Gödel shocked the world by proving that [1] and [2] cannot both be true:

  Either you can prove false statements or there are true statements that are not provable.

- Most people believe that mathematics is incomplete rather than mathematics is inconsistent

- Turing was one of the people who showed that [3] is false by his work on Turing machines and the halting problem

- Important: mathematics is open in the sense that there cannot be a finite set of axioms and rules from which all mathematical truths can be proved.

Lahenduvus: Gödel

- Oluline tähelepanek Gödeli tulemustest: matemaatika on ses mõttes lahtine, et

  ei saa olla lõplikku aksioomide ja reeglite kogu, millest saab järeldada kõiki tegelikult õigeid matemaatikaväiteid.

  Juba ainult täisarvudega tegelev matemaatika on lahtine.

- Aga: aksioome ja reegleid saab alati lisada. Iga õige matemaatikaväide on järeldatav, kui sul on piisav hulk aksioome ja reegleid.

- Aksioomid ja reeglid ise on asjad, mida me ei saa tõestada: peame neid lihtsalt – mingil põhjusel – uskuma.

Lahenduvus: Gödel

Gödeli mittetäielikkuse teoreemi veidi nõrgem vorm:

„**A complete, consistent and sound axiomatization of all statements about natural numbers is unachievable„**

Sellel vormil on suhteliselt lihtne tõestus, mis kasutab universaalse peatumiskontrolli võimatust:

Assume that we have a consistent and complete axiomatization of all true first-order logic statements about natural numbers. Then we can build an algorithm that enumerates all these statements i.e. an algorithm N that, given a natural number n, computes a true first-order logic statement about natural numbers, such that for all the true statements there is at least one n such that N(n) is equal to that statement.

...

Lahenduvus: Gödel

...

Now suppose we want to decide whether the algorithm with representation a halts on input i. By using Kleene's T predicate, we can express the statement "a halts on input i" as a statement H(a, i) in the language of arithmetic. Since the axiomatization is complete it follows that either there is an n such that N(n) = H(a, i) or there is an n' such that N(n') = Ā¬ H(a, i). So if we iterate over all n until we either find H(a, i) or its negation, we will always halt.

This means that this gives us an algorithm to decide the halting problem. Since we know that there cannot be such an algorithm, it follows that the assumption that there is a consistent and complete axiomatization of all true first-order logic statements about natural numbers must be false.